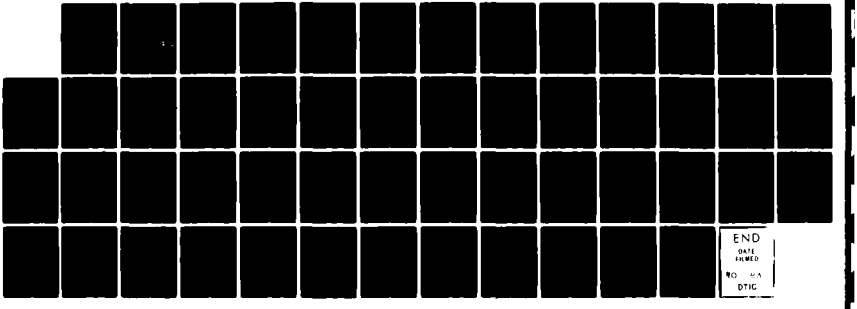
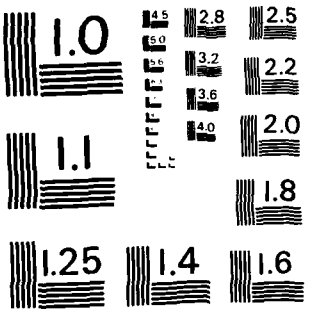
AD-A132 681 A STUDY OF SOFTWARE FAILURES AND RECOVERY IN THE MVS 1/1
OPERATING SYSTEM(U) STANFORD UNIV CA CENTER FOR
RELIABLE COMPUTING P VELARDI ET AL. JUL 83 CRC-TR-83-7
UNCLASSIFIED ARO-18690.6-EL DAAG29-82-K-0105 F/G 9/2 NL





Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ARO 18690.6-EL

AD-A132 692

## REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: CRC Tech. Rpt. 83-7
2. GOVT ACCESSION NO.:
3. RECIPIENT'S CATALOG NUMBER:

4. TITLE (and Subtitle): A Study of Software Failures and Recovery in the MVS Operating System
5. TYPE OF REPORT & PERIOD COVERED: Interim Tech. Report
6. PERFORMING ORG. REPORT NUMBER:

7. AUTHOR(s): Paola Velardi and Ravishankar K. Iyer
8. CONTRACT OR GRANT NUMBER(s): ARO DAAG-29-82-K-0105

9. PERFORMING ORGANIZATION NAME AND ADDRESS:
Center for Reliable Computing
Computer Systems Laboratory
Stanford University, Stanford, CA 94305

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: DD Form 2222, Project No. P-18690-EL

11. CONTROLLING OFFICE NAME AND ADDRESS:
U. S. Army Research Office
Post Office Box 12211
Research Triangle Park, NC 27709

12. REPORT DATE: July 1983
13. NUMBER OF PAGES: 50

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office):
Mr. James W. Gault
Electronis Division
U.S. Army Research Office
P. O. Box 12211, Research Triangle Park, NC 27709

15. SECURITY CLASS. (of this report): Unclassified
15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:

16. DISTRIBUTION STATEMENT (of this Report):
Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report):
N/A

DTIC ELECTE SEP 20 1983 S D A

18. SUPPLEMENTARY NOTES:
THE VIEW, OPINIONS, AND/OR FINDINGS CONTAINED IN THIS REPORT ARE THOSE OF THE AUTHOR(S) AND SHOULD NOT BE CONSTRUED AS AN OFFICIAL DEPARTMENT OF THE ARMY POSITION, POLICY, OR DECISION, UNLESS SO DESIGNATED BY OTHER DOCUMENTATION.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number):
Software reliability, fault tolerance, recovery, statistical analysis

20. ABSTRACT (Continue on reverse side if necessary and identify by block number):
This paper describes an analysis of system-detected software errors on the MVS operating system at the Center for Information Technology (CIT), at Stanford University. The analysis determines the most common error categories and relates them to the program in execution at the time of the error. The severity of the error is measured by evaluating the criticality of the program for continued system operation. The system recovery and error correction features are then analyzed and an estimate of the system fault tolerance to errors of different levels of severity is made.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Center for Reliable Computing

# A Study of Software Failures and Recovery in the MVS Operating System

Paola Velardi and Ravishankar K. Iyer

CRC Technical Report No. 83-7

(CSL TN No. 83-226)

July 1983

CENTER FOR RELIABLE COMPUTING
Computer Systems Laboratory
Departments of Electrical Engineering and Computer Science
Stanford University
Stanford, California 94305

This work was supported in part by the Department of the Army under Contract Number DAAG29-82-K-0105, the Ugo Bordoni Foundation and the Italian National Research Council (CNR).

# A Study of Software Failures and Recovery in the MVS Operating System

Paola Velardi and Ravishankar K. Iyer

CRC Technical Report No. 83-7

(CSL TN No. 83-226)

July 1983

CENTER FOR RELIABLE COMPUTING
Computer Systems Laboratory
Departments of Electrical Engineering and Computer Science
Stanford University
Stanford, California 94305

## ABSTRACT

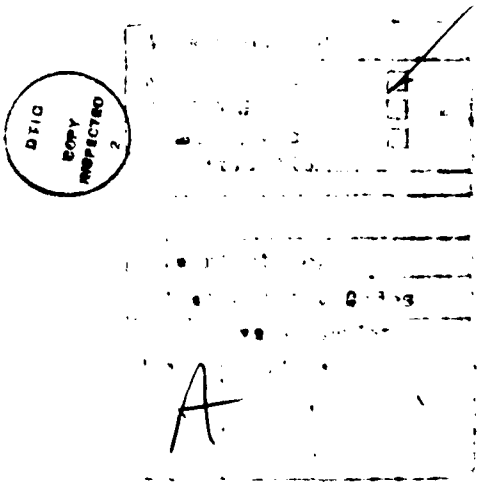
This paper describes an analysis of system-detected software errors on the MVS operating system at the Center for Information Technology (CIT), at Stanford University. The analysis determines the most common error categories and relates them to the program in execution at the time of the error. The severity of the error is measured by evaluating the criticality of the program for continued system operation. The system recovery and error correction features are then analysed and an estimate of the system fault tolerance to errors of different levels of severity is made.

Keywords: Software reliability, fault tolerance, recovery, statistical analysis

CONTENTS

# FIGURES





# TABLES

# 1. INTRODUCTION

The design of reliable and fault tolerant software systems is one of the most important issues facing computer designers today. Computer software cost and reliability are the major problem areas affecting sophisticated systems. An important reason for this is that the processes affecting a software system are highly complex and interactive. Theoretical models can only deal with a limited range of problems. It is, most often problems outside the range of these models that cause the most severe malfunctions. Thus, there is no better substitute, at this stage, for results based on actual measurements and experimentation [Curtis 80]. Such studies however are few and far between [Denning 80].

This paper presents results of one such analysis conducted on the MVS operating system on the IBM 3081 at the Center for Information Technology (CIT) at Stanford University. CIT is the main campus computation facility. It is used for production programs (payrolls and administration), student and research projects, and for general purpose computing. The installation consists of two IBM 3081 processors which run the MVS operating system. The two processors are loosely coupled, e.g., they have distinct control programs and different I/O configurations. On a typical day, the two systems support around 500 users and run approximately 4000 batch jobs.

The general objective of this study was to determine the causes of software errors (recoverable and non-recoverable) in a fully operational production environment. We wished to understand the software problems and attributes that affect the correct handling of adverse system conditions. The data on error detection and recovery is automatically logged

by the operating system. Thus it was possible to investigate not only the general question of software reliability but also examine the automatic system recovery features. In particular the following points were addressed:

1. What are the most common types of software errors in a fully operational environment and their relative frequencies?
2. What is the impact of the errors on the system?
3. How robust is the system in recovering from different types of errors?
4. What inferences can be drawn from the analysis in relation to the system fault tolerance and recovery features?

The approach adopted was to start with a substantial quantity of high quality data on all software errors, (recoverable and non-recoverable). An error collection mechanism which selected and filtered the raw data so as to cluster records referring to the same error, was developed. An analysis of the clustered data was then performed to determine the most common types of system errors and their effect on system integrity. Finally, the system fault tolerance and recovery features were evaluated by measuring their ability to successfully recover from system problems of various levels of severity.

In our analysis we differentiate between the terms "error" and "failure". A failure is a software "error" which causes the termination of the system (i.e., a system failure). Thus an error, in general, may or may not result in a failure.

Before describing this work in detail, an overview of related research in this area is presented.

## 2. RELATED RESEARCH AND MOTIVATION

Designing hardware systems that tolerate faults is relatively well understood, at least from a theoretical viewpoint. However, the problem of software fault tolerance has yet to be well investigated [Hecht 80a,b]. A reason for this is that neither the error generation process nor the prediction problem are easy to comprehend, although the SIFT studies have been an important contribution [Wensley 78], [Melliar-Smith 81].

The term "software reliability model" is usually taken to mean mathematical models for assessing the reliability of software (in terms of statistical parameters such as Mean Time Between Failures) during the development, debugging or testing phases. A few of these models have also been applied in follow-up operational phases. Several competing models have appeared in the literature [Musa 1980], and a number of authors have attempted to analyse their suitability. An appreciation of the extent and nature of this discussion can be obtained from [Goel 80]. The main difficulty with these approaches is that, although each model appears to be valid within its own assumptions, there is insufficient experimental evidence available for its large scale validity.

Research most closely related to the present study is in the area of analysis of errors and their causes in large software systems. [Endres 75] discusses and categorises errors and error frequencies during the internal testing phase of the IBM DOS/VS system. In [Thayer 78] data collected from four large software development projects is analysed. [Hamilton 78] applies the well known execution time model [Musa 80] to measure the operational reliability of computer center software, and

[Glass 80] examines the occurrence of persistent bugs and their causes in operational software. Another useful study is [Maxwell 78], which tabulates and examines error statistics on software.

None of these studies tries to relate system reliability or the error frequencies to the usage environment of the software itself in a systematic manner. Results based on such measurements are essential in order to evaluate the system fault tolerance and automatic recovery features. The argument for adopting a particular approach is more convincing if backed by experiments demonstrating its usefulness.

In an early study of failures at the SLAC (Stanford Linear Accelerator Center) computation facility, [Butner 80] and [Iyer 82a] found a strong correlation between the occurrence of failures and the level of system activity at the time of failure. A more detailed and accurate analysis of failures on a VM/370 system (in service at SLAC since February 1981) confirmed this relationship [Rossetti 82].

The operational phase of mature software is somewhat different from the development, debugging, and testing phases. A typical situation is one where frequent changes and updates are installed either by the installation programmers or by the vendor. Often the vendor will install a change, to fix an error found at some other installation, without any notification to the installation management. In a sense the system being measured represents an aggregate of all such systems maintained by the vendor.

It is clear that more experimental studies on operational software are necessary before a sound analytical basis can be developed for software reliability evaluation. The MVS system on the IBM 3081 at CIT pro-

vided an ideal opportunity in this regard. The operating system automatically collects information on error detection and correction. The state of the machine at the time of the error is also recorded. The following section provides a overview of the error handling and collection mechanism on MVS.

## 3. SOFTWARE ERROR MEASUREMENT: HOW IT WORKS

This study concentrates on all software errors detected by the operating system. In addition to error detection, the operating system also provides for error correction through either retry, or a job or task termination. There is a close relation between the hardware and the operating system in this area. An overview of software error detection and recovery appears in [IBM 80]; detailed descriptions may be found in [IBM 81] and [IBM 79].

There are four different types of software problems for which a software record is generated:

1. Program check - indicates a hardware detected software error.
2. Invalid supervisor call - indicates a supervisor service request issued by an unauthorized program.
3. Program abnormal termination (ABEND) - indicates an abnormal termination (by the system or by the program itself) of the executing program.
4. Restart key depressed - indicates an operator detected anomaly and is caused by a restart operation.

These events cause the normal execution of the program to be interrupted, and the control to be transfered to a system control program

called the Recovery Termination Manager (RTM). Specifically, the RTM supervises the error handling process either by giving control to a recovery routine or by itself providing for error recovery in the absence of a recovery routine. In either case recovery can be in the form of a job or task termination or a retry. A task is a module or a sub-program of a job.

There are two types of recovery routines:

1. Functional Recovery Routines (FRR): These are provided in MVS for critical system programs.
2. Task Recovery Routines: These may be written for critical user and subsystem programs using an MVS facility called Extended Subtask Abend Exit (ESTAE).

At the end of the recovery process, the RTM invokes the error recording routines to generate a software record of the incident. The data set containing this information is called SYS1.LOGREC. Generally, FRR routines will write a record to LOGREC while ESTAE routines will generate a record only if the programmer has specifically provided for this in the routine.

A standard header on the software record consists of a time-stamp, the model and serial number of the CPU where the error occurred, and the name of the jobstep involved. In addition, the record contains data on the type of detection, the event causing the detection and the detailed symptom of the error. Data describing the steps in the recovery process is also provided. A sample of the error data is given in Fig. 1.

At CIT, software records have been systematically generated and archived since March 1982. The next section describes the processing performed on the raw data to make it usable for analysis.

| OBS | TIMESTMP | JOB | STYPE | CPU | CPUMODEL | EVENT | SYMPTOM | RECNAME | RESULT | JOBTERM |
|---|---|---|---|---|---|---|---|---|---|---|
| 319 | 15APR82:14:13:45 | CS76FRD2 | MVDSYSERR | 220162 | 3081 | PROGCHK | A00000 | IECVXFRR | CONTTERM | NO |
| 320 | 15APR82:14:13:52 | CS76PRB2 | MVDSYSERR | 20162 | 3081 | PROGCHK | A00000 | IECVXFRR | CONTTERM | NO |
| 321 | 15APR82:14:14:12 | CDLBP132 | MVDSYSERR | 220162 | 3081 | PROGCHK | A00000 | IECVXFRR | CONTTERM | NO |
| 322 | 15APR82:14:14:13 | CDLCP132 | MVDSYSERR | 220162 | 3081 | PROGCHK | A00000 | IECVXFRR | CONTTERM | NO |
| 323 | 15APR82:14:14:13 | CDLBP132 | MVDSYSERR | 220162 | 3081 | PROGCHK | A00000 | IECVXFRR | CONTTERM | NO |
| 324 | 15APR82:14:14:13 | INIT | MVDSYSERR | 220162 | 3081 | PROGCHK | A00000 | | CONTTERM | NO |
| 325 | 15APR82:14:14:13 | INIT | MVDSYSERR | 220162 | 3081 | PROGCHK | A00000 | | CONTTERM | NO |
| 326 | 15APR82:14:14:15 | PROBS | MVDSYSERR | 220162 | 3081 | PROGCHK | A00000 | IECVXFRR | CONTTERM | NO |
| 327 | 15APR82:14:14:27 | MSTRJCL | MVDSYSERR | 220162 | 3081 | PROGCHK | 0C1000 | | CONTTERM | NO |
| 328 | 15APR82:14:14:28 | ES4TAX | MVDSYSERR | 20162 | 3081 | PROGCHK | A00000 | IECVXFRR | CONTTERM | NO |
| 329 | 15APR82:14:14:32 | MSTPJCL | MVDSYSERR | 220162 | 3081 | PROGCHK | 0C1000 | IEESB665 | RETRY | NO |
| 330 | 15APR82:14:14:32 | MSTRJCL | MVDSYSERR | 220162 | 3081 | PROGCHK | 0C1000 | IEESB665 | CONTTERM | NO |
| 331 | 15APR82:14:14:34 | MSTRJCL | MVDSYSERR | 20162 | 3081 | PROGCHK | 0C1000 | | RETRY | NO |
| 332 | 15APR82:14:14:56 | DBS | MVDSYSERR | 20162 | 3081 | PROGCHK | A00000 | IECVXFRR | CONTTERM | NO |
| 333 | 15APR82:14:15:11 | DSAZFUND | MVDSYSERR | 220162 | 3081 | PROGCHK | A00000 | IECVXFRR | CONTTERM | NO |
| 334 | 15APR82:14:15:11 | INIT | MVDSYSERR | 220162 | 3081 | PROGCHK | A00000 | | CONTTERM | NO |
| 335 | 15APR82:14:15:21 | DBS | MVDSYSERR | 20162 | 3081 | PROGCHK | A00000 | IECVXFRR | CONTTERM | NO |
| 336 | 15APR82:14:15:27 | MSTRJCL | MVDSYSERR | 20162 | 3081 | PROGCHK | 0C1000 | IEESB665 | CONTTERM | NO |
| 337 | 15APR82:14:15:27 | MSTRJCL | MVDSYSERR | 220162 | 3081 | PROGCHK | 0C1000 | IEESB665 | RETRY | NO |
| 338 | 15APR82:14:15:29 | MSTRJCL | MVDSYSERR | 20162 | 3081 | PROGCHK | 0C1000 | | RETRY | NO |
| 339 | 15APR82:14:15:42 | MSTRJCL | MVDSYSERR | 220162 | 3081 | PROGCHK | 0C1000 | | CONTTERM | NO |
| 340 | 15APR82:14:16:00 | MSTRJCL | MVDSYSERR | 20162 | 3081 | PROGCHK | 0C1000 | IEESD665 | CONTTERM | NO |
| 341 | 15APR82:14:16:00 | MSTRJCL | MVDSYSERR | 220162 | 3081 | PROGCHK | 0C1000 | IEESB665 | RETRY | NO |
| 342 | 15APR82:14:16:02 | MSTRJCL | MVDSYSERR | 220162 | 3081 | PROGCHK | 0C1000 | | RETRY | NO |
| 343 | 15APR82:14:16:04 | MILTEN | SVDSYSERR | 220162 | 3081 | ROUTSVC | 05C000 | IEFAB4E6 | RETRY | NO |
| 344 | 15APR82:14:16:04 | NET | MVDSYSERR | 220162 | 3081 | PROGCHK | 0C1000 | ISTAPCES | CONTTERM | NO |
| 345 | 15APR82:14:16:09 | MSTRJCL | MVDSYSERR | 220162 | 3081 | PROGCHK | 0C1000 | IEESD665 | RETRY | NO |
| 346 | 15APR82:14:16:09 | MSTRJCL | MVDSYSERR | 220162 | 3081 | PROGCHK | 0C1000 | IEESD665 | RETRY | NO |
| 347 | 15APR82:14:16:09 | MSTRJCL | MVDSYSERR | 220162 | 3081 | PROGCHK | 0C1000 | IEESD665 | CONTTERM | NO |
| 348 | 15APR82:14:16:10 | MSTRJCL | MVDSYSERR | 20162 | 3081 | PROGCHK | 0C1000 | IEESB665 | CONTTERM | NO |
| 349 | 15APR82:14:16:11 | DBS | MVDSYSERR | 20162 | 3081 | PROGCHK | A00000 | IECVXFRR | CONTTERM | NO |
| 350 | 15APR82:14:16:12 | MSTPJCL | MVDSYSERR | 20162 | 3081 | PROGCHK | 0C1000 | | RETRY | NO |
| 351 | 15APR82:14:16:12 | MSTRJCL | MVDSYSERR | 220162 | 3081 | PROGCHK | 0C1000 | | RETRY | NO |
| 352 | 15APR82:14:16:16 | ORVGGSER | MVDSYSERR | 220162 | 3081 | PROGCHK | A00000 | IECVXFRR | CONTTERM | NO |
| 353 | 15APR82:14:16:17 | INIT | MVDSYSERR | 220162 | 3081 | PROGCHK | A00000 | | CONTTERM | NO |

Figure 1: Sample of software errors

## 4. BUILDING THE DATA BASE

An objective of this project was to make data management as automatic as possible so that it is unnecessary to know the particulars of operating systems, software monitors, record formats, and the like. The Statistical Analysis System (hereafter called SAS) [SAS 79] provided, in addition to its procedures for statistical analysis, a rich environment for data handling. Initially, the raw data set (which is in hexadecimal code) was compacted in order to extract the relevant information, and to provide explanations for hexadecimal codes. Then, the records believed to be repeated occurrences of the same problem (referred to as error re-occurrence) were clustered. The result of this manipulation was a data set ready to be fed into statistical analysis programs.

## 4.1 PROCESSING THE SOFTWARE ERROR DATA

A collection of SAS programs performs the following steps:

(i) Select, Decode and Classify: The raw LOGREC data includes CPU, channel, and device errors for all equipment in the installation. Only the software records on the two IBM 3081's were selected for this analysis. In each software record there are a number of bits describing the type of error, its severity, and the result of hardware and software attempts to recover from the problem. The general software error status indicators provided by the hardware and software are TYPE (of detection), EVENT (causing the detection) and SYMPTOM (code or symptom of the error).[1]

(ii) Sort By Processor and Time: To facilitate clustering in the next step it was necessary to sort the data by CPU id (serial number) and time of error within CPU id.

(iii) Cluster: Clustering of recurrent errors was performed for two reasons:

1. to obtain error distributions and statistics unbiased by error re-occurrence.
2. to measure the extent of the re-occurrence phenomenon.

Figure 1 contains an example of a recursive error. Consider observations 329 through 331: the problem coded 0C1 (an operation code exception) occurred while a task of the supervisor program MSTRJCL was executing. This error was detected by the hardware, and generated a

---

[1] The IBM names for these fields are [IBM 79]: TYPE - HDRTYP; EVENT - SDWERRA; SYMPTOM - SDWACMPC.

program check. The routines which handled the problem were unable to recover, and the task affected by the error was terminated. Sometime later the problem re-occurred (obs. 336 - 342, 345 - 348, 350 - 351, etc).

The example shows that an error cannot always be isolated in a single step. The programmer or the operator might be unable to diagnose the problem, and simply decide to rerun the program. Error recurrence might also be due to an error that does not cause the termination of the program. In this case, no action is taken by the system, and the problem might recur later (for example, if it is a pattern sensitive error). Devising a suitable clustering policy was an important step of this work. A careful analysis of the data and extensive experimentation was necessary.

The clustering algorithm analyses the data and merges observations that satisfy either one of the following conditions:

1. The observations have exactly the same time stamp. This was done to cluster records generated by the recurrence of the same error. Note that more than one record can be generated for the same incident if more than one recovery routine is specified for the process. This is called percolation [IBM 81].
2. The errors recur within a short time interval of each other, and have the same SYMPTOM code. This was done to capture repetitions of the same error. The determination of time interval to be used involved a careful analysis of the data and some experimentation. Finally, a time interval of 15 minutes was found to be most appropriate.

The clustering algorithm employs three steps. The unclustered data set is first sorted by time and CPU id. The observations that satisfy the conditions above are merged into a single record. These steps, however, fail to merge observations which satisfy conditions 1 and 2, but are interleaved with records referring to an apparently different problem (different SYMPTOM code). An example of this was illustrated in Fig. 1. To get rid of this problem, observations that satisfy condition 2, but are not consecutive, are also merged into a single cluster.

The result is a set of clustered errors. Associated with each cluster is information consisting of error classifications, number of points in the cluster (NPOINTS), time of the first (FIRST) and last (LAST) error in the cluster, the length of the cluster (SPAN = LAST - FIRST), the time between clustered errors (TBE), and a variety of status data provided by the hardware and operating system.

Some interesting things can be learned from a cursory analysis of the clusters derived as stated. Summary statistics for the number of points in a cluster (NPOINTS) and time spanned by a cluster (SPAN) are shown below in Figure 2. Figure 2 shows clearly that the clustering algorithm is having an effect by gathering long bursts of errors into a few large clusters, indicated by the maximum 340 points and 4497 seconds time

Original Errors: 7,197  Clustered Errors: 1562

| | NPOINTS | SPAN (seconds) |
|---|---|---|
| Mean | 4.60 | 46.8 |
| Median | 1 | 0.0 |
| 90th Percentile | 6 | 51.1 |
| Minimum | 1 | 0.0 |
| Maximum | 340 | 4497.0 |

Percentage bar charts

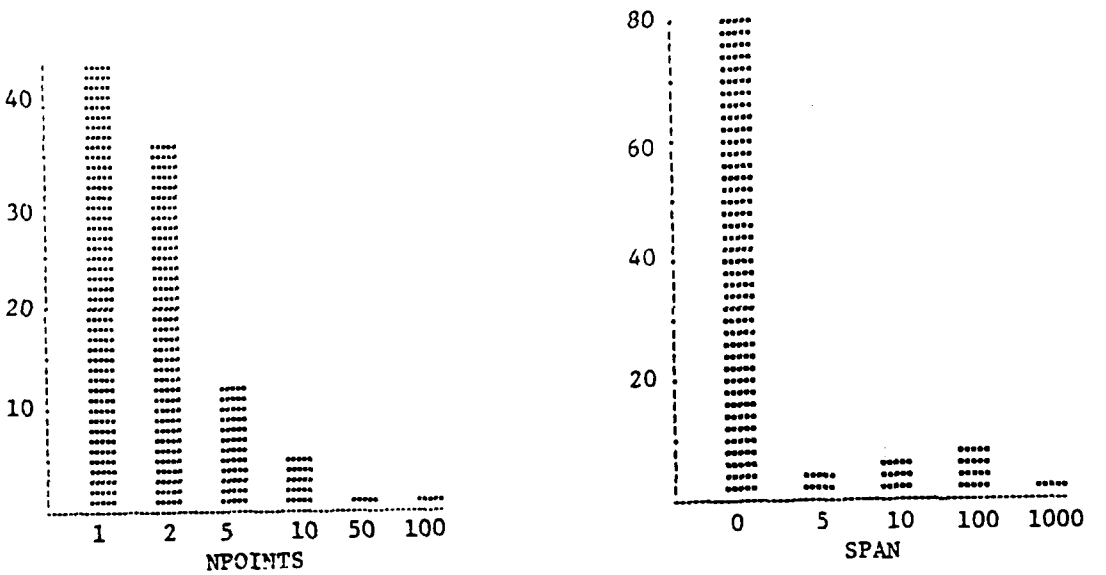



Figure 2: Clustering statistics

span. In fact, only 19% of the observations cluster more than two software records, and only 5% cluster more than 10 records. Large clusters (more than 100 records) account for just 0.7%. As far as the time length of the incident is concerned, about 82% of the incidents last less than 5 seconds, 8% last more than 100 seconds, and less than 1%

have a time length greater than 1000 seconds. The table also shows that lone errors predominate, with median cluster size of one and time span of zero, showing that the clustering algorithm is not artificially forcing them together. The accompanying bar charts also show this behavior. Clustering is important in error analysis to avoid biasing the results with repeated errors from the same failing component.

## 5. BASIC STATISTICS

This section presents a preliminary analysis of the data. The time period has been divided in two parts:

1. EARLY period: from March 1982 through May 1982.
2. LATE period: from June 1982 through April 1983.

The reason for this is that the system configuration was changed in June 1982, and a corresponding change was noticed in the error data. In the EARLY period only one CPU was active, while in the LATE period the system load was shared by two CPUs.

Initially two charts were generated. The chart in Fig. 3 gives the number of clustered software records for each month. During the first three months (from March to May 1982) MVS went into production on a 3081 CPU, which processed the workload previously supported by two IBM 3033 CPUs. The number of terminals managed by a single CPU almost doubled. Even though this workload was compatible with the new system specifications, a noticeably large number of problems occurred on the system. In June 1982, a second 3081 was added. The number of reported errors was considerably lower after June.

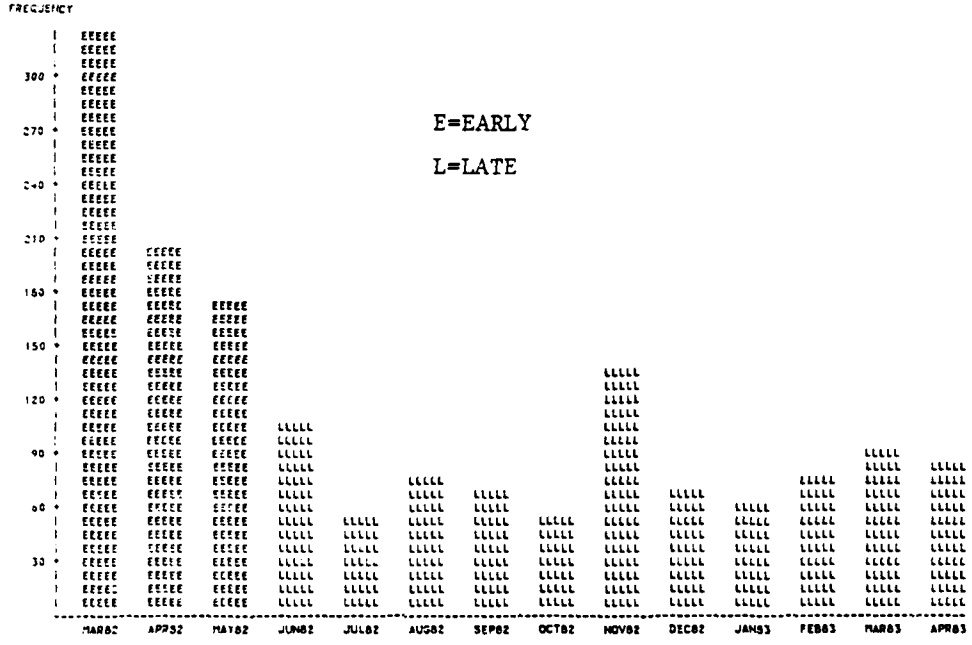



Figure 3: Frequency plot of software errors by month

Figure 4 shows a histogram of the percentage of software errors by the hour of day. The histogram has been obtained by averaging the daily observations over the entire period of study. This chart appears to follow rather closely the typical interactive load at CIT. It also compares quite favorably with a similar plot of software failures, obtained in a preceding study of the VM/370 system [Rossetti 82], which showed a strong relationship between software failures and interactive workload.

Table 1 provides more detailed statistics for the system. The table gives the mean, standard deviation, maximum and minimum value for the

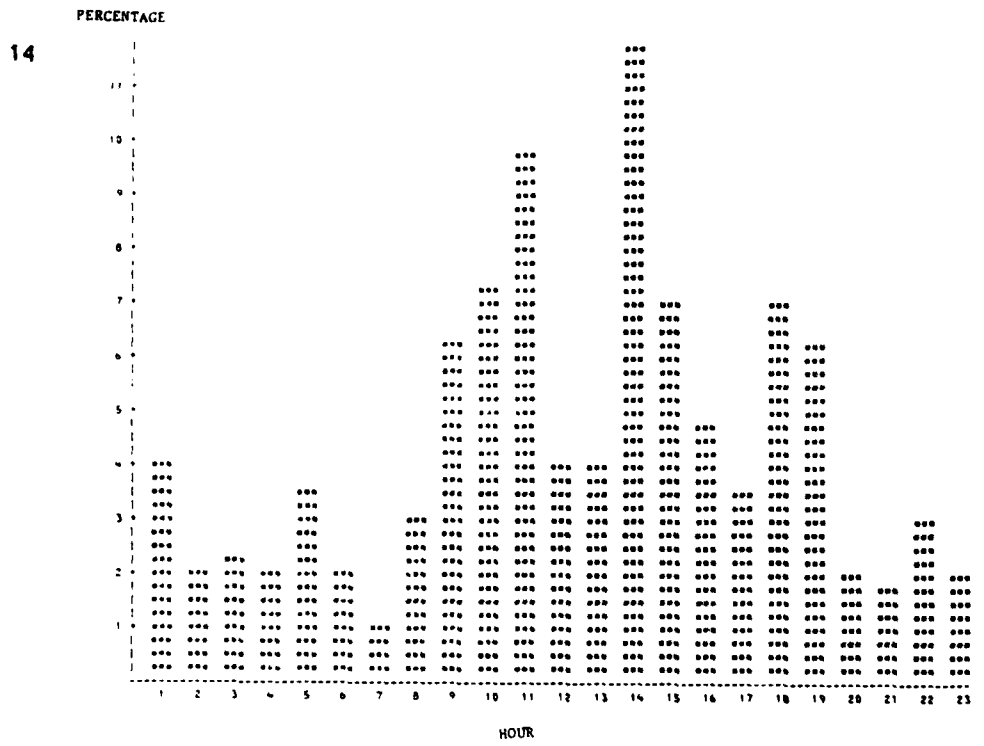



Figure 4: Hour of day plot of software errors

TABLE 1

Statistics on software errors

| Time Between Errors (Hours) | | | |
|---|---|---|---|
| | EARLY (3 months) | LATE (10 months) | COMBINED (13 months) |
| Mean | 7.5 | 24.4 | 17.2 |
| Standard deviation | 22.1 | 86.6 | 67.6 |
| Minimum | 0.0 | 0.0 | 0.0 |
| Maximum | 358.0 | 1704.0 | 1704.0 |
| Median | 1.8 | 5.8 | 3.2 |
| 90th Percentile | 18.4 | 48.5 | 34.7 |

time between clustered errors (TBE). Note the dramatic improvement in the mean time between errors, between the EARLY and LATE periods.

Table 2 provides an analysis of the various error detection mechanisms. It is found that software was the main detection mechanism (53%), followed by hardware (32%).

The next two sections investigate the nature of the errors and their impact on the system. Section 6 provides statistics on error categories and on recovery actions taken by the system. Section 7 relates these errors to the job in execution at the time of the error.

TABLE 2

Type of error detection

| | EARLY | | LATE | | COMBINED |
|---|---|---|---|---|---|
| TYPE | Freq. | % | Freq. | % | % |
| S/W Detected | 391 | 56.2 | 438 | 50.6 | 53.1 |
| H/W Detected | 192 | 27.6 | 315 | 36.4 | 32.5 |
| Lost Record | 110 | 15.8 | 112 | 12.9 | 14.2 |
| Operator Detected | 2 | 0.3 | 1 | 0.1 | 0.2 |

# 6. ANALYSIS OF THE DATA

This section investigates the nature of the errors and how they are handled by the system. In particular, the following questions are considered:

1. What are the most common error categories?
2. How good is the system at recovering from different types of errors?
3. What is the impact of errors on the system?

Six error categories were defined. To each category, a measure of its severity was ascribed, and the frequency of its occurrence was determined. Finally, for each category the success or failure of the recovery process was assessed.

## 6.1 ERROR CLASSIFICATION

In common with other analyses of this type, the error SYMPTOM codes provided by the system were grouped into classes of similar problems. The error classes were chosen by consulting with the CIT staff to determine commonly encountered problems. In addition, other studies of this nature were also consulted (e.g., [Thayer 78], [Endres 75], [Rossetti 82]). Finally, it was important to make sure that each error category had a statistically significant number of errors in it.

Six classes of errors were defined:

1. Control: indicates the invalid use of control statements and invalid supervisor calls.
2. I/O and data management: indicates a problem occurred during I/O management or during the creation and processing of data sets.

3. Storage management: indicates an error in the storage allocation/de-allocation process or in virtual memory mapping.
4. Storage exceptions: indicates addressing of non-existent or inaccessible memory locations.
5. Programming exceptions: indicates a program error other than a storage exception.
6. Deadlocks: indicates a system or operator detected endless loop, endless wait state or violation of system or user defined time limits.

Data tagged "LOSTRECS" (Lost Records) were purged from the data set. These accounted for 15% of the total. In addition, 1.3% of the records had invalid or missing SYMPTOM codes and were also purged.

The next subsection discusses the error data and determines the severity of each error category. The effect of the recovery process is subsequently considered. Where significant, the two periods of study are separately analysed so as to determine the differences in error behaviour between employing one or two CPU's.

## 6.2 ERROR STATISTICS

There are significant differences in the error distributions between the two periods of study. As noted earlier the time between errors and the error frequency was considerably higher in the EARLY period (one overloaded CPU) than in the LATE period (two CPU's).

Table 3 gives the percentage distribution of the errors during the two analysed periods. On the average, the two major error categories are storage exceptions (26%) and storage management (28%). Storage

problems decreased in absolute terms but not in percentage terms. Monthly plots for these and other important error categories appear in the Appendix.

TABLE 3

*Distribution of error categories during the two periods*

| | EARLY | | LATE | | COMBINED |
|---|---|---|---|---|---|
| Error type | Freq. | % | Freq. | % | % |
| Storage management | 215 | 36.7 | 159 | 21.1 | 27.9 |
| Storage exceptions | 115 | 19.6 | 229 | 30.4 | 25.7 |
| Deadlocks | 77 | 13.1 | 207 | 27.5 | 21.2 |
| I/O and data management | 112 | 19.1 | 43 | 5.7 | 11.6 |
| Programming exceptions | 41 | 7.0 | 62 | 8.2 | 7.7 |
| Control | 16 | 2.7 | 47 | 6.2 | 4.7 |
| Invalid | 10 | 1.7 | 7 | 0.9 | 1.3 |
| ALL | 586 | 100.0 | 754 | 100.0 | 100.0 |

* Note that "LOSTRECS" have been purged from this data set.

Recall that a major feature of the MVS operating system is the multiple virtual storage organisation. Storage management is a high volume activity and is critical to the proper operation of the system. One might therefore expect its contribution to errors to be siginificant. Thus, even though the absolute number of storage problems goes down, the

fractional contribution remains high. Another reason for such a large percentage of storage problems under increased workload was due to mishandling of hardware failures. The error detection mechanism on MVS is not always able to diagnose software problems resulting from a hardware failure.[2]

It can be seen from the table that resource management (control, I/O and data management, and storage management) errors have significantly decreased in LATE period, both in absolute and percentage (58% vs. 33%) terms. The reduction in resource management problems in the EARLY period is believed to be related to the reduction in workload per CPU resulting from the introduction of the second CPU. This is also substantiated by the results, on the relationship between system activity and software failures, reported in [Rossetti 82]. A possible explanation is as follows: when the number of jobsteps handled by a single CPU increases, the virtual and real storage management activity correspondingly increases. A typical problem that occurs under these circumstances is in synchronization between routines, handling common data or control blocks. Examples of control blocks are the descriptors of program address spaces. With increased levels of interactive loading, complex states can occur, for which the integrity and consistency of such control blocks are not guaranteed, and a failure can occur.

It is also observed that deadlocks and exceptions increased in the second period in percentage terms. In absolute values deadlocks and exceptions do not exhibit a substantial difference between the two periods. These are in fact problems related to the program itself and not

[2] We believe this problem of hardware/software interface needs to be studied in detail and will be the subject of another paper.

to the complexity of the workload. The Appendix provides plots for various error categories on a monthly basis.

## 6.3 ANALYSIS OF RECOVERY MANAGEMENT

The question addressed in this section is to determine which types of errors the system is able to handle, and how. The recovery process on MVS is intended to isolate an error in order to preserve the system from damage. Functional recovery routines (FRR's) are provided for important system functions [Auslander 81]. Thus we expect that, the more severe the problem, as far as the system resources and control structures are concerned, the more likely it is that a recovery routine is provided. Recovery of a program, e.g., the correction of the error is possible only under certain circumstances. A frequent example is the occurrence of an unpredicted state. In this case, re-establishing a suitable environment and issuing a retry can get rid of the problem. In other cases the system will try to recover by terminating the task or even the job in progress.

In order to evaluate the effectiveness of recovery management on MVS, we commenced by investigating when recovery routines are specified and why. Table 4 shows that recovery routines were provided for nearly 70% of the reported problems. The percentage is more than 85% for deadlocks and for storage and data management. For storage exceptions however this percentage drops to 35.2%.

TABLE 4

Provision of recovery routines by error type

| Error type | Frequency | Percent |
|---|---|---|
| I/O and data management | 142 | 91.6 |
| Storage management | 323 | 86.4 |
| Deadlocks | 244 | 85.9 |
| Control | 41 | 65.1 |
| Programming exceptions | 48 | 46.6 |
| Storage exceptions | 121 | 35.2 |
| ALL | 933 | 69.6 |

Table 5 provides an analysis of system recovery management. Overall, the percentage of successful retries is 36% of all errors. Task terminations account for nearly 52% of all errors and job terminations for 12%. The percentage of retry is highest (65%) for storage management errors.[3] Exceptions lead frequently to a task termination, even though this does not imply termination of the job. Deadlocks result in a substantially large number of job and task terminations (94% combined) and only 6% retries. A program that exceeds its timing constraints or that which indefinitely holds system resources is almost always terminated. In fact, further processing could seriously damage the system.

[3] This seems to indicate that such problems are often due to the occurrence of a particular, unpredicted state. In this case, a simple retry can be successful.

TABLE 5

Effectiveness of the recovery by error type

| Error Type | JOBTERM % | TASKTERM % | RETRY % |
|---|---|---|---|
| Storage management | 0.3 | 34.5 | 65.2 |
| Control | 0.0 | 42.9 | 57.1 |
| I/O and data management | 1.3 | 55.5 | 43.2 |
| Storage exception | 0.6 | 73.8 | 25.6 |
| Programming exception | 0.0 | 83.5 | 16.5 |
| Deadlocks | 55.3 | 38.7 | 6.0 |
| All | 12.2 | 51.9 | 35.9 |

Job or task terminations are potentially severe problems. The severity of the problem is best determined by what the system was doing at the time of the error. Thus, in order to truly evaluate the effectiveness of system recovery management, it is important to determine the type and criticality of the job in control at the time of the error. The following section analyses the type of job affected by a reported error.

## 7. RELATING ERROR TO SYSTEM FUNCTION

This section relates the errors and the result of the recovery process with the programs affected by the error. It is obvious that the termination of a job that performs a critical system function has a much more severe impact on the system than a user or production job. In order to understand this problem the following analysis was performed in close cooperation with the CIT systems management staff.

1. Job Categories: Using the job name (at error occurrence) provided by the system, four job categories were defined, viz. Supervisor (MVS) related jobs, Interactive jobs, Testing jobs, and User/Production jobs.
2. Job Functions: These categories were further divided in three groups of job functions: Critical (for system survival), Essential (would degrade but not crash the system) and Non-Essential (to system survival). Most of the MVS and interactive jobs fell in the critical group.

Interactive and Supervisor jobs are each involved in nearly 17% of all errors. User and production account for 37% of the problems; testing jobs occur in 12% of the cases. No job names could be attached to nearly 22% of the reported errors (unclassified).[4] Detailed tabulations are given in the Appendix. It was noted that error involvement of MVS and interactive jobs fell from 53% in the EARLY period to around 18% in the LATE period. This compares favourably with the reduction in resource management problems with the introduction of the second CPU.

[4] In these cases the job name field was missing or the error was reported by a functional recovery routine, and the name of the source problem program was not available or the name of the job could not be recognised.

The error categories were found to be uniformly distributed over both the job categories and the job functions (see Appendix for tables). The exceptions were "deadlocks", which mainly occurred on non-essential jobs (84%) and I/O and data management which occurred mainly on critical jobs (77%). Table 6 evaluates the effectiveness of the recovery routines in dealing with critical, essential and non-essential jobs. Retries occured on 43% of errors involving critical jobs and for 68% on essential jobs. The worst figure for job terminations was on non-essential jobs (23%).

Importantly, in over 50% of the cases where system critical jobs are involved, task termination results. The task is a module of the critical job and usually system termination (recall that this is defined as a failure) results. Similar, though slightly improved figures are found for essential jobs. This points toward an inadequacy in recovery management, since one would like better recovery and far less task terminations when critical and essential jobs are involved.

Table 6 also shows that recovery routines were specified in about 65% of the errors where critical jobs were involved. In interpreting this table recall that recovery is possible even when no recovery routine is provided through the Recovery Termination Manager. The percentage of failures in cases where recovery routines were specified is 44% versus 80% when no recovery routine was specified. This appears to show the FRR's and ESTAE's are having an effect in improving the system fault tolerance but there is still considerable scope for improvement. For essential jobs (where we expect degradation in service but not necessarily a system failure) the percentage of failures where recovery routines

TABLE 6

Recovery management

| Job Criticality and Type of Recovery | | | | |
|---|---|---|---|---|
| | RETRY % | TASKTERM % | JOBTERM % | Frequency |
| Critical | 43.3 | 53.0 | 3.7 | 402 |
| Essential | 68.6 | 23.5 | 7.8 | 51 |
| Non-Essential | 24.8 | 51.9 | 23.3 | 592 |

| Effectiveness of Recovery Routines | | | |
|---|---|---|---|
| | Recovery Routines Provided % | Failures (Recovery Routines Provided) % | Failures (Recovery Routines Not Provided) % |
| Critical | 65.7 | 44.3 | 80.4 |
| Essential | 78.4 | 20.0 | 72.7 |

are specified drops to nearly 20% versus 72% where no recovery routines are specified. Thus the recovery routines are doing a much better job in dealing with essential than with critical jobs. In fact one would like these figures to be reversed.

TABLE 7

Effect of recovery routines for critical jobs

| Error Type | Freq. | Rcvy Routine Provided % | Failures** (Rcvy Routine Provided) % | Failures** (Rcvy Routine Not Provided) % |
|---|---|---|---|---|
| Control | 22 | 63.6 | 21.4 | 100.0* |
| Deadlocks | 29 | 82.8 | 100.0 | 100.0 |
| I/O and data management | 74 | 82.4 | 90.2 | 7.7 |
| Storage management | 161 | 79.5 | 7.8 | 81.8 |
| Storage exceptions | 82 | 18.3 | 46.7 | 92.5 |
| Programming exceptions | 31 | 64.5 | 80.0 | 63.6 |
| All | 402 | 65.7 | 44.3 | 80.4 |

* The number of observations was not statistically significant (<=4)
** Failure: Job or task termination on a critical job

Table 7 relates the provision of recovery routines to the specified error categories when critical jobs are involved (i.e., potentially serious system problems). It is found that recovery routines are most effective in dealing with storage management problems (an important feature of MVS). When no recovery routines are provided, the probability of a storage management failure is high (81%). The recovery routines

are weakest in dealing with deadlocks, I/O and data management[5] problems and programming exceptions. Thus it would appear that these are the particularly vulnerable areas of the system where further attention could be directed. In order to quantify the above figures we define and evaluate measures of fault tolerance and reliability in the following section.

## 8. MEASURES OF SYSTEM FAULT TOLERANCE AND RELIABILITY

This section evaluates the following measures using the collected data.

1. The mean time for error handling (MTEH) is defined as:

$$MTEH = \frac{\Sigma\ SPAN(i)}{N} \qquad (1)$$

where:

SPAN(i) Length of Cluster i

N Number of Clustered Errors

2. The average (MFR) failure rate:

$$MFR = \frac{\text{Number of Failures}}{\text{Time Period of Measurement}} \qquad (2)$$

where:
Number of Failures = No. of job/task terminations[6] of critical jobs

---

[5] Notice that, for I/O and data management, we have a strange situation in that the system fails over 90% of the time when recovery routines were provided versus only 7.7% when no recovery routine was provided. This was caused by a particular bug. The RTM was able to successfully retry and no recovery routine was involved. These retries have considerably biased the statistics for this error category.

[6] Recall that the termination of a critical job or its task (module) almost always results in system failure and is defined as such.

3. The software fault tolerance (FT) (i.e., the probability of recovery given that a software error has occurred):

$$FT = 1 - \frac{\text{Number of Failures}}{\text{Total Number of Errors}} \qquad (3)$$

To obtain more detailed information, these measures were evaluated for each error category and for the two periods (where significant). The results are shown in Tables 8, 9 and 10.

Table 8 on MTEH shows how quickly the system is able to handle an error. The result of the error handling process could be a successful recovery or a termination (failure). The generally larger handling times for critical jobs reflect the fact that not only is the system attempting to isolate the error but also is trying to avoid termination. Very long error handling times occur on storage exceptions (large clusters). It was found that many of these errors were due to hardware problems. Refer to the Appendix for tabulations.

Table 9 calculates the failure rate for each error category. The average system failure rate was found to be 11.1 per 1000 hours in the LATE period. Looking at the failure rate figures we notice a general improvement between the two periods over all error categories. In the EARLY period a high failure rate exists for management and storage problems. In the LATE period storage exceptions appear to be the dominating error category. In general the failure rate ranges between 1.2 and 3.4 per 1000 hours.

TABLE 8

Mean time for error handling (Seconds)

| | EARLY | LATE | COMBINED |
|---|---|---|---|
| Critical | 38.0 | 46.2 | 40.4 |
| Essential | 4.8 | 4.2 | 4.6 |
| Non-Essential | 13.8 | 40.0 | 26.4 |

TABLE 9

Average failure rate (Failures/1000 Hours)

| Error type | EARLY | LATE |
|---|---|---|
| Control | 2.2 | 0.8 |
| Deadlocks | 4.5 | 2.6 |
| I/O and data management | 21.0 | 1.1 |
| Storage management | 9.9 | 2.0 |
| Storage exceptions | 19.9 | 3.4 |
| Programming exceptions | 6.3 | 1.2 |
| All | 63.8 | 11.1 |

Table 10 presents the system fault tolerance under two conditions. It shows how well the system handles all problems i.e., regardless of

the type of job in control at the time of the error (all jobs). In order to quantify how well the system recovery management handles serious system problems, the fault tolerance measure (FT) was evaluated, given that a critical job was in control at the time of the error. The overall system fault tolerance to a software error is found to be 0.88. The table shows that the system is weak in dealing with errors occurring on critical jobs. This calculation was also performed for each error category. It is seen that the system deals best with storage management and control problems. It is at its weakest in dealing with deadlocks and exceptions. The figure for I/O and data management is also rather low. As expected the results match with the recovery management analysis of the previous section.

TABLE 10

Fault tolerance

| Error Type | All Jobs | Critical Jobs |
|---|---|---|
| Control | 0.80 | 0.50 |
| Deadlocks | 0.90 | 0.00 |
| I/O and data management | 0.42 | 0.24 |
| Storage management | 0.89 | 0.77 |
| Storage exceptions | 0.63 | 0.16 |
| Programming exceptions | 0.69 | 0.26 |
| All | 0.88 | 0.43 |

## 9. CONCLUSIONS

It has been the purpose of this paper to present an analysis of software related errors on the MVS operating system at CIT. Storage management and storage exceptions were found to be the major error categories. This was related to the criticality of the storage management activity on the MVS system. The decreasing contribution, of these and other categories of errors, was found to be related to the changes to the system during the analysed periods, especially the reduction of workload per CPU through the addition of a second CPU. The occurrence of software errors closely matched the interactive workload on the system. This compares quite favourably with the results on VM/370 reported in [Rossetti 82].

Data on the recovery process showed that recovery routines are provided on MVS mainly for those problems which affect some system resource. The fact that very few large error clusters occur shows that the recovery process is reasonably successful in isolating a problem. The large clusters that do occur are mainly due to the fact that the error detection on MVS is not always able to diagnose software problems resulting from a hardware failure.

The effectiveness of error recovery was measured with regard to its intended purpose, i.e., to avoid system damage. The error severity (and the effectiveness of recovery management) was evaluated by relating the error occurrence to the type of program affected by the error. Data on error recovery showed that the system fault tolerance almost doubles when recovery routines are provided, in comparison with the case where no recovery routines are available. The system recovery routines are

most effective in handling storage management problems (an important feature of MVS). However, even when recovery routines are provided, there is almost a 50% chance of system failure when critical system jobs are involved. Thus there is still considerable scope for improvement. Deadlocks, I/O and data management and exceptions are the main problem areas. Deadlocks in MVS, it is felt, are best only be dealt with through improved error detection. Here, both pre-checking and post-checking of data structures could be a possibility. In the other cases more robust recovery routines and better handling of hardware errors could produce a substantial improvement. It should be noted however that some caution is advised in applying the statistical figures calculated here to other situations.

Importantly, the results obtained in this paper demonstrate that it is possible to derive quantitative measures for system fault tolerance and recovery management. This information can be very valuable in pinpointing major problem areas where further work, oriented toward enhanced recovery management, can be directed. It is suggested that other systems be measured and analysed in this manner so that a wide spectrum of practical results on operational software are available.

## 10. ACKNOWLEDGMENTS

The authors would like to thank Prof. E.J. McCluskey for his interest in this work and for extensive discussions during the period of this study. Special thanks are extended to Larry Rivers and Lincoln Ong at CIT for providing valuable insight into the MVS system; to D.J. Rossetti, A. Mahmood and Dr. D.J. Lu for their careful reading of an early draft of this paper.

This work was supported in part by the U.S. Army Research Office under contract number DAAG29-82-K-0105, the Ugo Bordoni Foundation and the Italian National Research Council (CNR). The views, opinions, and/or findings contained in this document are those of the authors and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other official documentation.

## REFERENCES

[Auslander 81] M.A. Auslander, D.C. Larkin and A.L. Scherr, "The evolution of the MVS operating system", IBM Journal of Research Development , Vol. 25, No. 5, September 1981

[Butner 80] S.E. Butner and R.K. Iyer, "A statistical study of reliability and system load at SLAC", Digest, Tenth International Symposium on Fault Tolerant Computing, October 1980.

[Curtis 80] B. Curtis, "Measurement and experimentation in software engineering", Proceedings of the IEEE, Vol. 68, No. 9, pp. 1144-1157, September 1980.

[Denning 80] P.J. Denning, "On learning how to predict", Proceedings of the IEEE, Vol. 68, No. 9, pp. 1099-1103, September 1980.

[Endres 75] A. Endres, "An analysis of errors and their causes in systems programs", IEEE Trans. Software Engineering, Vol. SE-1, No. 2, pp. 140-149, June 1975.

[Glass 80] R.L. Glass, "Persistent software errors", IEEE Trans. Software Engineering, Vol. SE-7, No. 2, pp. 162-168, March 1981.

[Goel 80] A.K. Goel, "A summary of the discussion on 'An analysis of competing software reliability models'", IEEE Trans. Software Engineering, Vol. SE-6, No. 5, pp. 501-502, September 1980.

[Hamilton 78] P.A. Hamilton and J.D. Musa, "Measuring reliability of computation center software", Proc. Third Int. Conf. Software Engineering, Atlanta Georgia, pp. 29-36, May 1978.

[IBM 81] IBM Corp., OS/VS2 MVS, System Programming Library: MVS Diagnostics Techniques, Order No. GC28-0725, 1981.

[IBM 80] IBM Corp., OS/VS2 MVS, System Programming Library: Supervisor, Order No. GC28-1046, 1980.

[IBM 79] IBM Corp., OS/VS2 MVS, System Programming Library: SYS1.LOGREC Error Recording, Order No. GC28-0677-5, 1979.

[Hecht 80a] H. Hecht, "Current issues in fault tolerent software", Proceedings COMPSAC 80, Chicago Illinois, pp. 603-607, November 1980.

[Hecht 80b] H. Hecht, "Mini-tutorial on software reliability", Proceedings COMPSAC 80, Chicago Illinois, pp. 383-385, November 1980.

[Iyer 82a] R. K. Iyer, S. E. Butner, and E. J. McCluskey, "A statistical failure/load relationship; Results of a multi-computer study," IEEE Transactions on Computers, July 1982.

[Iyer 82b] R.K. Iyer and D.J. Rossetti, "A statistical load dependency model for CPU errors at SLAC," The Dig. FTCS-12, Twelfth International Symposium on Fault Tolerant Computing, Santa Monica California, June 1982.

[Littlewood 80] B. Littlewood, "Theories of software reliability: How good are they and how can they be improved?", IEEE Trans. Software Engineering, Vol. SE-6, No. 5, pp. 489-500, September 1980.

[Maxwell 78] F.D. Maxwell, The determination of measures of software reliability, Final Report, NASA-CR-158960, The Aerospace Corporation, El Segundo California, December 1978.

[Melliar-Smith 81] P.M. Melliar-Smith and R.L. Schwartz, "Current progress on the proof of SIFT," The Dig. FTCS-11, Eleventh International Symposium on Fault Tolerant Computing, Portland, Maine, June 1981.

[Musa 80] J. Musa, "The Measurement and management of software reliability", Proc. IEEE, Vol. 68, pp. 1131-1143, September 1980.

[Rossetti 82] D.J. Rossetti and R.K. Iyer, "Software related failures on the IBM 3081: A relationship with system utilization", Proc. COMPSAC 82, Chicago Illinois, November 82.

[SAS 79] SAS User's Guide, 1979 Edition., Sas Institute Inc., Raleigh, North Carolina, 1979.

[Thayer 78] T.A. Thayer, M. Lipow and E.C. Nelson, Software Reliability: Study of Large Project Reality, TRW Series of Software Technology, Vol. 2, North-Holland, 1978.

[Wensley 78] J. Wensley, et. al., "SIFT: Design and analysis of a fault tolerant computer for aircraft control," Proc. IEEE, Vol. 66, No. 10, pp. 1240-1254, October 1978.

# Appendix A

## TABLES AND PLOTS

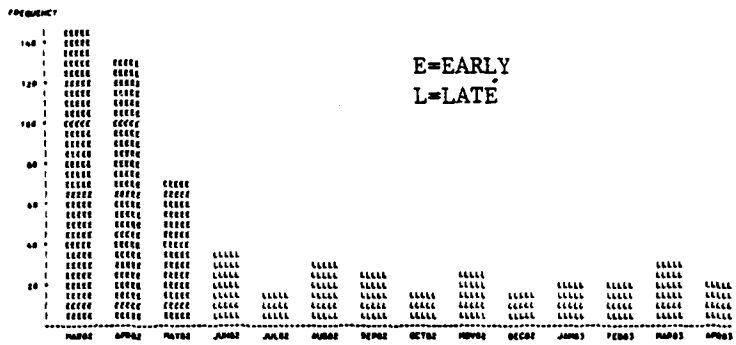



Figure 5: Frequency plot of resource management errors by month

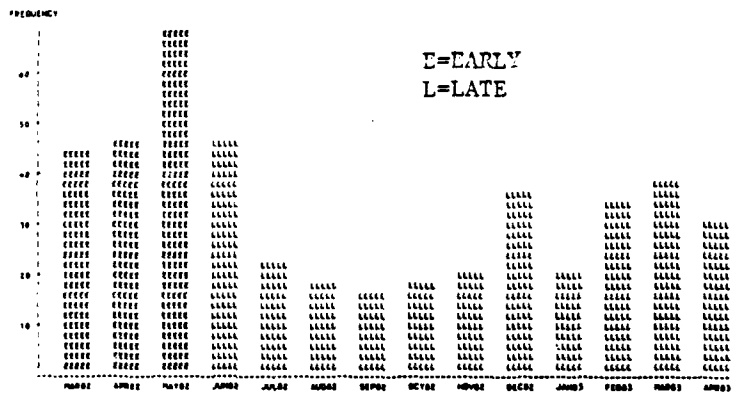



Figure 6: Frequency plot of storage and programming exceptions by month

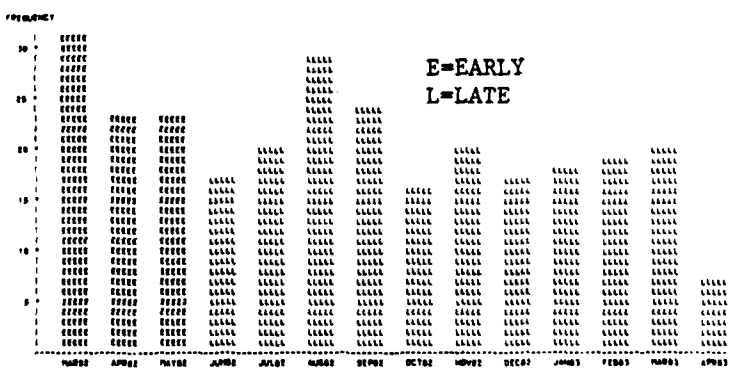



Figure 7: Frequency plot of deadlocks by month

TABLE 11

Job classification

| | EARLY | | LATE | | COMBINED | |
|---|---|---|---|---|---|---|
| Job Type | Freq. | % | Freq. | % | Freq. | % |
| Interactive System | 189 | 32.3 | 49 | 6.5 | 238 | 17.7 |
| MVS System programs | 124 | 21.2 | 91 | 12.1 | 215 | 16.0 |
| System Testing | 37 | 6.3 | 128 | 17.0 | 165 | 12.3 |
| User and Production | 121 | 20.7 | 306 | 40.6 | 427 | 31.9 |
| Unclass/Invalid | 115 | 19.7 | 180 | 23.9 | 295 | 22.0 |

TABLE 12

Job functions and type of error

| Error Type | Critical | | Essential | | Non-Essential | |
|---|---|---|---|---|---|---|
| | Freq. | % | Freq. | % | Freq. | % |
| Control | 22 | 40.7 | 7 | 13.0 | 25 | 46.3 |
| Deadlocks | 29 | 10.6 | 10 | 3.6 | 236 | 85.8 |
| I/O and data management | 74 | 77.1 | 1 | 1.0 | 21 | 21.9 |
| Storage management | 161 | 46.8 | 4 | 1.2 | 179 | 52.0 |
| Storage exceptions | 82 | 43.6 | 13 | 6.9 | 93 | 49.5 |
| Programming exceptions | 31 | 42.5 | 5 | 6.9 | 37 | 50.7 |
| All | 402 | 38.5 | 51 | 4.9 | 592 | 56.7 |

TABLE 13

Mean time for error handling (Seconds)

| Error Type | Critical | | Essential | | Non-Essential | |
|---|---|---|---|---|---|---|
| | EARLY | LATE | EARLY | LATE | EARLY | LATE |
| Control | 16.3 | 0.0 | 7.5 | 3.8* | No Obs. | 127.6 |
| Deadlocks | 0.6 | 39.7 | 0.0* | 0.0* | 20.6 | 21.9 |
| I/O and data management | 17.5 | 17.5 | 0.0* | No Obs. | 53.7 | 31.0 |
| Storage management | 29.2 | 35.4 | 40.3* | 0.0* | 1.9 | 42.8 |
| Storage exceptions | 106.9 | 108.6 | 0.0? | 4.6 | 0.8 | 62.2 |
| Programming exceptions | 15.5 | 2.0 | 0.0* | 13.0* | 0.0* | 83.6 |
| All | 38.0 | 46.2 | 4.8 | 4.2 | 13.8 | 50.6 |

* The number of observations was not statistically significant (<=4)

TABLE 14

Fault tolerance

| Error Type | Critical | | Essential | |
|---|---|---|---|---|
| | EARLY | TOTAL | EARLY | TOTAL |
| Control | 0.54 | 0.45 | 0.50* | 1.00* |
| Deadlocks | 0.00 | 0.00 | 0.25 | 0.25 |
| I/O and data management | 0.08 | 0.64 | 0.00* | 0.00* |
| Storage management | 0.84 | 0.35 | 1.00* | 1.00* |
| Storage exceptions | 0.12 | 0.22 | 0.75 | 0.78 |
| Programming exceptions | 0.33 | 0.10 | 0.67* | 0.50* |

* The number of observations was not statistically significant (<=4)

# Appendix B

# MVS ERROR DETECTION AND RECOVERY PROCESSING

## B.1 ERROR DETECTION

The supervisor in MVS offers many services to detect and process abnormal conditions during system execution.

1. The hardware detects conditions such as memory violations, program errors (arithmetic exceptions, invalid operation codes) and addressing errors.
2. The software also provides detection of software problems.

   The data management and supervisor routines ensure that valid data are processed and non-conflicting requests are made. Examples are the incorrect specification of a parameter in a control structure or in a system macro, or a supervisor call issued by an unauthorized program.

   The installation might improve the system error detection capability by means of a software facility called Resource Access Control Facility (RACF). The RACF is used to build detailed 'profiles' of system software modules. These profiles are defined in order to inspect the correct usage of system resources.

   The user might also define his own detection mechanisms by means of the Set Program Interruption Element (SPIE) macro. This macro instruction detects programmer defined exceptions like

using an incorrect address or attempting to execute privileged instructions.

3. The operator might detect some evident error condition and decide to cancel or restart the job. For example, the operator can detect loop conditions or endless wait states.

A software record also contains the information about the event (EVENT) that caused the record to be generated, and a 12 bit symptom code (SYMPTOM) describing the reason for the program abnormal termination. This codes are issued by the system or by the problem program that used an ABEND macro instruction. The system and user completion codes appear together in the SYMPTOM field. User codes are meaningful only for specific applications.

Table 15 describe the values assumed by the variable EVENT. Table 16 gives some example of common system SYMPTOM codes encountered in this study. The detection mechanism and the action taken by the system are also described. More than 500 different SYMPTOM codes are issued by the system for a problem program.

TABLE 15

Event that caused program termination

| Variable EVENT<br>Values | Meaning |
|---|---|
| MACHECK | A hardware event caused a machine check that could not handle the problem |
| PROGCHECK | A program check interrupt occurred due to the detection of some exception or to the violation of some memory protection mechanism |
| TRSFAIL | A translation error, e.g., an error occurred during the storage allocation process |
| RESTART | The operator pressed the restart key |
| ROUTABT | A system service routine detected an invalid SVC and issued an abnormal termination of the program (ABEND) |
| ROUTSVC | A system routine issued an invalid supervisor call (SVC) |
| PROGABT | The program itself requested the ABEND |
| SYSABT | The system detected a problem and forced a program ABEND |

TABLE 16

Examples of ABEND reason codes

| Hex code | Explanation | System action |
|---|---|---|
| 05A | A service routine that handles real storage deallocation received an invalid address | The program that called the service routine or the routine abnormally terminates |
| 071 | The operator determined that the program was in a loop or endless wait state | The operator pressed the RESTART key |
| 0C1 | Operation exception: an operation code is not assigned | A program interruption occurred; the task is terminated if no routine had been specified to handle the interruption |
| 020 | The error occurred during the creation of a data set due to the incorrect specification of some data parameter | The task is terminated if no routine has been specified for the problem program |

## B.2 RECOVERY PROCESSING

Whenever a program is abnormally interrupted due to the detection of an error, the Supervisor gets control. If the problem is such that a further processing could degrade the system or destroy data, the Supervisor gives control to the Recovery Termination Manager (RTM). If a recovery routine is available for the problem program, RTM gives control to this routine before processing the program termination.

Recovery is designed as a means by which the system can prevent total loss. The purpose of a recovery routine is to free the resouces kept by

the failing program (if any), to locate the error and to request either for a continuation of the termination process or for a retry. Recovery routines are generally provided to cover all MVS functions [Auslander 81]. It is however the responsibility of the installation or of the user to write recovery routine for other programs.

More than one recovery routine can be specified for the same program; if the latest recovery routine asks for a termination of the program, the RTM can give control to another recovery routine (if provided). This process is called 'percolation'.

The percolation process ends if either a routine issues a valid retry request, or no more routines are available. In the latter case, the program and its related subtasks are terminated. The termination of a program might imply the termination of jobstep. If a valid retry is requested, a retry routine restore a valid status, using the information supplied by the recovery routine(s), and can give control to the program. In order for a retry to be valid the system should verify that there is no risk of recurrence of the error to the same recovery routine, and that the retry address is properly specified. Figure 8 illustrates the steps in the recovery process.

Traces of the recovery process are recorded on LOGREC. This includes the name and the type of the recovery routine which handled the problem (RECNAME), the result (RESULT) of the recovery process and the impact of the error on the related jobstep (JOBTERM). A description of these fields is given in Table 17. Other data collected during the recovery process, includes detailed program status information such as the contents of registers and the program address space identifier. This can be helpful in error diagnosis.

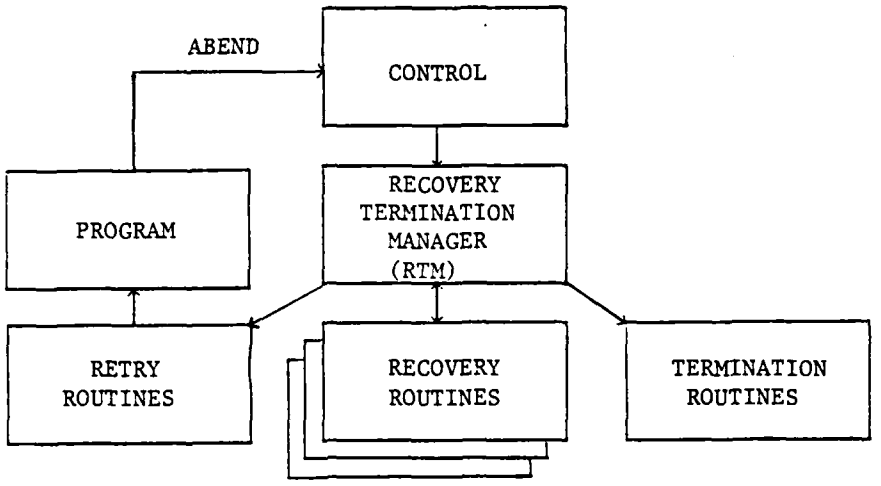



Figure 8: Software handling of software errors on MVS

TABLE 17

Data on the recovery process

| Variable name | Values | Meaning |
|---|---|---|
| RECNAME | 8 character name | Name of the recovery routine which handled the problem |
| RESULT | RETRY | The recovey routine decide that a retry might be successful |
| | CONTTERM | The recovery routine asks to continue with termination (this might imply percolation) |
| JOBTERM | YES/NO | If JOBTERM=YES the entire jobstep has to be terminated |

During the recovery process the system basically attempts to maintain operation despite an error. It is possible that the recovery process itself encounters the same error. In this case, there exists the risk of recursive recovery processes, or the generation of bad data. However, such occurences can be detected by analyzing the SDWA field into LOGREC. If the jobname for example is 'NONE-FRR', this indicates that the record is generated by a functional recovery routine during a recovery attempt. Finally, if the recording process was also affected by an error, a LOSTREC value appears in the TYPE field.

maintain
process
the risk
. How-
eld into
tes that
a recov-
ed by an

END

DATE
FILMED

10 — 83

DTIC